Analyse abstraite de missions sous PILOT

Damien Massé
EA 3883, Université de Bretagne Occidentale, Brest
damien.masse@univ-brest.fr

Résumé

Nous étudions la possibilité de réaliser un analyseur par interprétation abstraite de plans de contrôle robotique écrits dans le langage PILOT. L'objectif de cette analyse est de prouver, à partir d'une modélisation virtuelle de l'environnement et du comportement du robot, des propriétés valides sur toutes les exécutions possibles du plan. La mise au point de l'analyseur débute par une définition d'une sémantique de traces de PILOT, qui permet d'identifier en particulier les problèmes que posent les aspects temporels et le parallélisme du langage. Nous étudions ensuite les abstractions génériques utilisables sur cette sémantique, les possibilités de contourner les problèmes spécifiques au langage, et le plan d'un éventuel analyseur.

Mots-clés : robotique, PILOT, analyse statique.

1 Introduction

PILOT est un langage et un environnement de programmation pour le contrôle distant de systèmes robotiques, basé sur la notion d'*action*. Un programme (ou *plan*) est écrit en PILOT dans un environnement graphique qui permet également de simuler une exécution ou de modifier le plan en cours.

Notre objectif est d'améliorer cet environnement en y ajoutant un analyseur abstrait qui dériverait des propriétés démontrées sur le plan dessiné, à partir d'un modèle virtuel de l'environnement du robot. Cet environnement, ainsi que les actions du robot elles-mêmes, étant non déterministe, notre analyse se veut exhaustive, dans le sens où les propriétés générées doivent être vérifiées pour toute exécution, ce qui ne peut être fait par simulation. La théorie de l'interprétation abstraite permet de mettre au point de telles analyses, à partir d'une sémantique opérationnelle formelle du langage de programmation. Dans une première étape, nous définissons donc une sémantique de traces de PILOT, en dégageant le domaine concret utilisé qui servira de base à la mise au point de l'abstraction. Nous verrons tout particulièrement le problème posé par l'aspect temporel des actions, et la présence du parallélisme dans le programme.

Ensuite, nous travaillerons sur l'abstraction elle-même. L'objectif sera de mettre au point une analyse générique efficace (en terme de complexité) et suffisamment expressive, qui pourra être instanciée avec des domaines numériques abstraits comme les polyèdres ou les octogones. Nous exposerons enfin les problèmes rencontrés lors de l'implantation en cours de l'analyseur, et son éventuelle intégration dans la plate-forme d'environnement.

2 Le langage PILOT

PILOT[LRFM97] est un langage graphique et interprété destiné au contrôle de systèmes distants, doté d'un système de contrôle permettant la construction, l'exécution et la supervision des programmes (ou plans). L'objectif de PILOT est de fournir un environnement simple pour une programmation et une exécution sûre de plan.

En tant que langage, PILOT est basé sur la notion d'*action*, instruction élémentaire qui provoque une action du système distant dans son environnement (par l'intermédiaire d'ordres sur le robot). PILOT distingue deux types d'actions : les actions élémentaires et les actions continues, l'action élémentaire étant conçue pour se terminer d'elle-même tandis que la fin de l'action continue est provoquée par une autre primitive du langage. Les actions disposent de préconditions et de règles de

surveillance (la satisfaction des préconditions étant nécessaire pour permettre l'exécution de l'action, tandis que la satisfaction d'une règle de surveillance provoque en général l'arrêt de l'action).

En plus des actions, PILOT comprend des structures de contrôle pour construire les plans :

- la *séquentialité* ;
- la *conditionnelle*, où les conditions sont testées successivement et au plus une branche est exécutée ;
- l'*itération*, qui peut être fixe ou conditionnelle ;
- le *parallélisme* et la *préemption*. Dans le parallélisme, l'exécution se termine lorsque toutes les séquences en parallèle s'achèvent, tandis que dans la préemption, l'exécution se termine dès qu'une séquence se termine.

3 Sémantique de traces, domaine concret

Dans cette section nous cherchons à modéliser une sémantique des plans sans prendre en compte l'interactivité avec l'environnement de PILOT. Dans cette optique, un comportement qui demande l'aide de l'opérateur (par exemple, lorsqu'une action ne peut pas être exécutée à cause de la violation d'une précondition) sera considéré comme un cas d'erreur.

Une sémantique opérationnelle de PILOT à base de règles de transitions a été développée précédemment [LRFM97]. Nous formalisons ici une autre sémantique plus adaptée au type d'analyse que nous souhaitons faire.

3.1 Traces

Intuitivement, l'exécution d'un plan peut se décrire comme une fonction qui donne l'état du robot et de son environnement en fonction du temps. Cette fonction sera appelée *trace d'exécution* du plan. Dans le cas de PILOT, l'évolution de l'environnement (incluant le robot lui-même) est continu (dans le sens où le temps s'écoule de façon continue), avec des changements d'états discrets (lorsqu'une action commence ou se termine). Une première définition de l'ensemble des traces $\mathcal{T}$ peut donc être :

$$\mathcal{T} = \mathbb{R}^+ \hookrightarrow \Sigma,$$

où Σ représente l'ensemble des états possibles du système (robot, environnement), et $\hookrightarrow$ représente une application partielle (la trace étant définie sur un intervalle $[0, T_{\text{fin}}]$.

Néanmoins, une première approximation conduit à considérer certains changements d'états comme *instantanés*, plus précisément l'ensemble des changements d'états dûs aux constructions de PILOT qui ne concernent pas les actions[1].

Dans ces conditions, plusieurs états peuvent apparaître à un même instant t, et notre ensemble de traces $\mathcal{T}$ devient une fonction des réels positifs vers des mots finis d'états :

$$\mathcal{T} = \mathbb{R}^+ \hookrightarrow \Sigma^+. \tag{1}$$

Le domaine sémantique concret est alors ;

$$\mathcal{S} = \wp(\mathcal{T}) = \wp\left(\mathbb{R}^+ \hookrightarrow \Sigma^+\right).$$

3.2 Définition de Σ, points de programme

On peut décrire *Sigma* à l'aide de points de programmes (qui décrivent l'état général du robot) et d'un environnement qui regroupe à la fois les variables internes du robot et la description de l'environnement externe (comprenant en particulier d'éventuelles valeurs sur les capteurs). En conséquence on définit Σ comme :

$$\Sigma = \wp(\mathcal{P}) \times (\mathcal{V} \hookrightarrow \text{Values}), \tag{2}$$

où $\mathcal{P}$ désigne l'ensemble des points de programme, $\mathcal{V}$ l'ensemble des variables et Values les valeurs (le plus souvent numériques) que peuvent prendre ces variables. Nous noterons $\mathcal{E}$ l'ensemble des environnements $\mathcal{V} \hookrightarrow \text{Values}$.

Du fait du parallélisme, un état comporte forcément plusieurs points de programmes. Nous pouvons différencier plusieurs « types » de points de programmes, qui correspondent à des différents types de transitions :

- des points « instables » $\mathcal{P}_I$, qui mènent à un autre point par une transition instantanée (entrée et fin de boucle, entrée de parallélisme, de condition...) ;

[1] Le temps réel de ces changements d'états, a priori très faible, dépend de l'implantation de PILOT et de l'architecture d'exécution du plan. Il devrait être borné par une valeur suffisamment petite par d'autres analyses

– des points d'« actions » $\mathcal{P}_A$ qui représentent une action en cours d'exécution, et dont l'évolution est temporelle (et peut dépendre des autres actions en cours) ;
– des points particuliers, agissant dans le cadre du parallélisme, en particulier des points de « synchronisation » en fin de séquences de parallélisme, et des points de « préemption », dont la transition instantanée provoque la fin des autres séquences. Nous regroupons ces points dans un ensemble $\mathcal{P}_S$.

Pour des raisons pratiques, on peut rajouter des points d'entrée $\mathcal{P}_E$ pour chaque action. On peut alors en déduire les conditions assurant une stabilité sur l'instant de l'état du robot : l'ensemble des points de programme courants doivent être dans $\mathcal{P}_A \cup \mathcal{P}_S$, avec un ensemble de conditions sur $\mathcal{P}_S$ indépendantes de l'évolution de l'environnement.

3.3 Sémantique des actions

Pour permettre la mise au point de l'analyseur, notre sémantique concrète est définie sous la forme d'un point fixe d'un opérateur. En particulier, il est nécessaire de mettre en dehors le caractère continue de l'écoulement du temps. Ainsi nous supposons l'existence d'une fonction qui donne, en fonction des actions en cours d'exécution, l'évolution de l'environnement jusqu'à la fin possible d'une des actions.

Une évolution (sans changement de points de programme) pourra alors être décrite par un triplet (t, e, P) où :
– $t \in \mathbb{R}^+$ est le temps d'exécution avant qu'une action s'arrête ;
– $e \in [0, t] \to \mathcal{E}$ donne l'évolution de l'environnement au cours de ce temps ;
– $P \subseteq \mathcal{P}_A$ est l'ensemble des actions qui se terminent au temps t.

Ainsi, nous supposons l'existence d'une fonction Act, prenant un ensemble d'actions repérées par leurs points de programme en paramètre (et éventuellement un réel associé à chaque action, donnant le temps depuis laquelle cette action est entamée), ainsi qu'un environnement initial, et renvoyant un ensemble d'évolutions décrits sous la forme précédente.

$$\begin{aligned} Act : \quad & (\mathcal{P}_A \hookrightarrow \mathbb{R}^+) \times \mathcal{E} \\ & \hookrightarrow \wp\left(\mathbb{R}^+ \times (\mathbb{R}^+ \hookrightarrow \mathcal{E}) \times \wp\left(\mathcal{P}_A\right)\right) \end{aligned} \tag{3}$$

La fonction Act, qui modélise les interactions du robot et de son environnement, est directement définie pour l'utilisateur de l'analyseur. Une méthode possible est de permettre à l'utilisateur de fixer, pour chaque action :
– des bornes sur la durée de l'action ;
– des préconditions et conditions de surveillance sur l'action ;
– des équations et inéquations différentielles (simples) sur l'environnement, représentant des contraintes lors de l'exécution de l'action.

Comme l'objectif est de vérifier des propriétés de sûreté sur le programme, il n'est pas nécessaire d'avoir une fonction Act exacte (donnant l'ensemble des solutions validant l'intersection des contraintes sur l'ensemble d'action), mais une version approchée surestimant l'ensemble des solutions.

3.4 Sémantique concrète

La fonction Act donne une transition fondamentale de notre sémantique qui se construit *en avant* (du passé vers le futur)[2]. Notre sémantique concrète est alors définie comme le plus petit point fixe (suivant un ordre adapté sur les traces) d'un opérateur ϕ sur l'ensemble $\mathcal{S}$:

$$[\![\text{Plan}]\!] = \text{lfp } \phi$$

Intuitivement, notre ensemble initial de traces partielles est constitué de traces réduite à l'état initial. ϕ étend ensuite chaque trace, soit par une transition instantanée lorsque cela est possible, ou par l'application de Act sur les actions courantes. Les traces arrivant à un état final (et éventuellement à un état d'erreur) sont conservées. Le plus petit point fixe de ϕ est alors la sémantique de notre programme.

4 Abstractions

Bien entendu, ni ϕ ni $[\![\text{Plan}]\!]$ ne sont calculables. Les techniques d'interprétation abstraite [Cou00] peuvent être utilisées pour mettre au point une approximation calculable de la sémantique. Cette mise au point passe par l'utilisation d'*abstractions*

[2] Il pourrait être intéressant ensuite de construire notre sémantique *en arrière*, à partir d'une fonction Act inverse.

du domaine concret. Nous présentons dans cette section le principe dans ce cas particulier.

4.1 Principe de l'interprétation abstraite

Notre sémantique concrète $[\![\text{Plan}]\!]$ s'exprime sous la forme du plus petit point fixe d'un opérateur ϕ sur un domaine ordonné $(\mathcal{S}, \sqsubseteq)$. Nous cherchons à déterminer une sémantique abstraite $[\![\text{Plan}]\!]^\sharp$ sur un domaine abstrait $(\mathcal{S}^\sharp, \sqsubseteq^\sharp)$ suffisamment expressif.

Nous définissons la *fonction de concrétisation* γ comme une fonction croissante de $\mathcal{S}^\sharp$ dans $\mathcal{S}$. Cette fonction associe à chaque élément abstrait l'ensemble des éléments concrets représentés[3]. . Nous construisons d'autre part une fonction $\phi^\sharp$, calculable, telle que, pour tout élément A de $\mathcal{S}^\sharp$:

$$\phi \circ \gamma(A) \ \subseteq \ \gamma \circ \phi^\sharp(A). \tag{4}$$

Autrement dit, $\phi^\sharp$ surapproxime dans l'abstrait le comportement de ϕ dans le concret. En général $\phi^\sharp$ peut se dériver de la définition de ϕ.

Alors :

$$\text{lfp } \phi \ \sqsubseteq \ \gamma(\text{lfp } \phi^\sharp). \tag{5}$$

En définissant la sémantique concrète $[\![\text{Plan}]\!]^\sharp = \text{lfp } \phi^\sharp$, cette équation nous garantit la sûreté de notre analyse.

4.2 Domaine abstrait

Notre domaine abstrait est défini par *abstractions successives*. Notre domaine final associera un ensemble d'environnements possibles pour chaque point de programme :

$$\mathcal{S}^\sharp = \mathcal{P} \rightarrow \mathcal{E}^\sharp. \tag{6}$$

L'environnement est étendu pour comprendre des compteurs temporels, à partir du début de la mission et à partir de certains points précis. La concrétisation associe alors l'ensemble des traces dont tous les états satisfont l'association entre points de programme et environnement.

Ici, $\mathcal{E}^\sharp$ est une abstraction de $\mathcal{E}$, indépendante de la structure interne de l'analyseur (de façon à pouvoir utiliser les domaines abstraits déjà existants).

[3]Pour des raisons de simplicité, nous supposons cette fonction croissante suivant les ordres $\sqsubseteq$ et $\sqsubseteq^\sharp$.

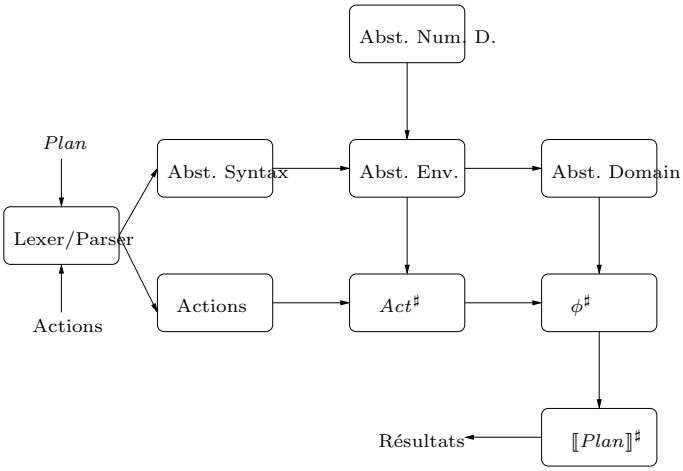

FIG. 1 – Plan de l'analyseur de PILOT.

La construction de la fonction de transfert abstrait ϕ peut se faire au cours de la construction des abstractions successives.

5 Implantation et intégration

Nous prévoyons d'implanter actuellement l'analyse en OCaML, langage adapté à la mise au points d'analyseurs de façon modulaire. L'analyseur sera décomposé en module selon le plan donné Fig 1.

6 Conclusion

Nous présentons la base d'une sémantique pour l'analyse statique de plans écrits en PILOT. Un analyseur à partir de cette sémantique est actuellement en cours d'écriture. Il sera nécessaire ensuite de l'interfacer avec l'environnement de développement PILOT et de l'adapter aux besoins des utilisateurs. La mise au point d'une analyse construite en arrière, ou une amélioration de la sémantique pour prendre en compte les probabilités d'apparition de certains événements, demanderaient encore des travaux théoriques supplémentaires.

Références

[Cou00] P. Cousot. Interprétation abstraite. *Technique et science informatique*, 19(1-2-3) :155–164, January 2000.

[LRFM97] E. Le Rest, J-L. Fleureau, and L. Marcé. Semantics and implementation of a language for telerobotics. Grenoble, France, 1997.